Insertion of a Random Task in a Schedule : a Real-Time Approach

Cyril Duron, Jean-Marie Proth

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Insertion of a random task in a schedule : a real-time approach.

Cyril DURON — Jean-Marie PROTH

N° 4193

Avril 2001

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4193--FR+ENG

Insertion of a random task in a schedule : a real-time approach.

Cyril DURON [*] [†] , Jean-Marie PROTH [‡†]

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Sagep

Rapport de recherche n° 4193 — Avril 2001 — 12 pages

Abstract: We consider the case of a single resource. A given schedule (possibly optimal) is evaluated by means of the sum of the delays of the tasks. A task appears in the system at a random time. The duration of this task is random, as well as its due date. The goal is to complete the task at the latest by its due date while increasing as little as possible the sum of the delays of the initial tasks. We have to find an algorithm that reduces as most as possible the amount of computation to be performed in real time at the expenses of the amount of computation to be performed off-line.

Key-words: Real-time, Scheduling, Single resource

[*] E-mail : duron@loria.fr
[†] Location : Inria / Sagep, UFR Scientifi que, Université de Metz, Ile du Saulcy, 57000 Metz, France.
[‡] E-mail : proth@loria.fr

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Insertion d’une tâche aléatoire dans un ordonnancement : Une approche temps-réel.

Résumé : Le problème qui suit concerne une ressource unique. Nous considérons un ordonnancement que nous pouvons supposer optimal, et dont le critère est la somme des retards par rapport à des délais connus. Une tâche aléatoire intervient à un instant donné, sa durée est connue, et elle doit être exécutée avant un délai impératif. L’objectif est de placer cette tâche en temps réel de manière à augmenter aussi peu que possible le critère attaché à l’ordonnancement initial. Tout le problème consiste à minimiser le volume des calculs à effectuer en temps réel et à traiter en différé les informations qui concernent l’ordonnancement initial, lequel est connu.

Mots-clés : Temps réel, Ordonnancement, Ressource unique

1 Introduction.

Predicability is of outmost importance for solving real-time scheduling problems. In this paper, we present a scheduling problem that concerns a single resource, which is a radar in this case. The goal is to minimize the sum of the delays of the tasks. Previous research work showed that algorithm EDF (Earliest Deadline First) [7], which schedules the tasks in the increasing order of their due dates, and the algorithm LL (last laxity) [1], are optimal when independant and preemptive tasks performed on a single processor are concerned. More precisely, all the tasks will be completed on time if such a solution exists. Unfortunately, preemption is possible when the resource is a processor, but it is no more possible in most of the other situations, and in particular when the resource is a radar or a machine.

When the tasks to schedule are non-preemptives, that is when a task cannot be suspended before completion if it has started, two types of real-time algorithms are usually considered, that is the algorithms dedicated to static problem and the algorithms dedicated to dynamic problems. A problem is said to be static when the tasks to be scheduled and the related constraints are known at the beginning of the process. The algorithms proposed in [6] and [3] are used in this case, when the real-time constraint applies. In the dynamic problems, the tasks enter the system randomly, and the next task when scheduling the current one may be known at the last instant only. The algorithms used to solve dynamic problems are of two types : the ones that apply when the resource capacity is able to perform the random tasks, and the others. In the first case, the EDF algorithm previously mentioned is optimal.

Another approach was recently proposed when the demands arrive randomly in a production system and must be scheduled immediatly, taking advantage of the idle time windows that have been let available by the demand previously scheduled. In this case, several resources are involved, and some of these resources may be able to perform the same operations. A real-time algorithm that aim at minimizing the makespan of the task under consideration has been proposed in [2], [4] and [5] for flow-shops, job-shops and assembly systems.

Let us come back to the simple resource case, and consider the case when the resource capacity is too small or may be too small to face the required production. In this environment, the efficiency of EDF decreases very fast, and we use the Spring algorithm [8] that proceed as the previous approach by scheduling each task as soon as it arrives in the system.

The problem we introduce in this paper is fundamentaly different from the previous ones. We consider that tasks that should be performed periodically are already scheduled, and that the sum of the delays of these tasks is known. In the case of a radar, these tasks are, for instance, the searching tasks. We consider that an unexpected task arrives in the system at a random time. For instance, an object is detected in the sky and a task should be performed as soon as possible in order to evaluate the object, or to decide if the signal detected is a noise or not. The goal is to start this task so as to fit with its due date while increasing as few as possible the sum of the delays of the tasks already scheduled. We define the problem in section 2. The basic relations are presented in section 3. The ways to use these relations to develop algorithms that perform most of the computation off-line are presented in section 4. Section 5 is dedicated to the presentation of two algorithms. Numerical results and the comparison between the two algorithms presented in section 5 are given in section 6. Section 7 is the conclusion.

2 Problem setting.

A schedule involving n tasks denoted by $a_1,\ldots,a_n$ is given. The time required to perform a_i is denoted by t_i for $i = 1,2,\ldots,n$. The starting times of these tasks are denoted by $\mu_1,\mu_2,\ldots,\mu_n$ respectively. Indeed, $\mu_i + t_i \leq \mu_{i+1}$ for $i = 1,2,\ldots,n-1$ since the radar performs at most one task at a time. The due dates of the tasks are denoted by $d_1,d_2,\ldots,d_n$ respectively. In the following part of this paper, we set :

$$\Delta_i = \mu_{i+1} - (\mu_i + t_i) \text{ for } i = 1,2,\ldots,n-1 \tag{1}$$

Δ_i is the period that starts at the end of task a_i and ends at the beginning of task a_{i+1}. Furthermore:

$$f_i = [d_i - (\mu_i + t_i)]^+ \text{ for } i = 1,2,\ldots,n-1 \tag{2}$$

Remember that $(a^+) = max(0,a)$ f_i represents the flexibility of task a_i, that is the amount of time a_i can be postponed without increasing the value of the sum C_n of the delays of the tasks, that is :

$$C_n = \sum_{i=1}^{n} (\mu_i + t_i - d_i)^+ \tag{3}$$

We want to insert in the initial schedule a task A that appears at time 0. Task A is defined by :

- its due date D that cannot be violated,

- its duration, denoted by θ

The goal is to minimize the increase of C_n after inserting the task A in the existing schedule.

3 Basic relations.

Assume that we want to insert task A after task a_k, $k = 1,\ldots,n-1$. In other words, A starts as soon as a_k is completed, that is at time $\mu_k + t_k$. Then, the increase of C_n resulting from the translation of a_{k+1} on the time axis is :

$$R_{k+1} = \begin{cases} 0 & \text{si} \quad \theta \leq \Delta_k \\ (\theta - \Delta_k - f_{k+1})^+ & \text{si} \quad \theta > \Delta_k \end{cases} \tag{4}$$

The explanation of (4) is straightforward. If $\theta \leq \Delta_k$, then task A can be inserted after task a_k and completed before a_{k+1} starts : the initial schedule is not modified. If $\theta > \Delta_k$, task a_{k+1} is delayed by $\theta - \Delta_k$. Then, if $\theta - \Delta_k \leq f_{k+1}$, the due date of a_{k+1} is not violated, and the increase of C_n is equal to zero, otherwise the increase of C_n is $\theta - \Delta_k - f_{k+1}$. This completes the explanation.

The increase of C_n resulting form the translation of a_{k+2} on the time axis is :

$$R_{k+2} = \begin{cases} 0 & \text{si} \quad \theta \leq \Delta_k + \Delta_{k+1} \\ (\theta - \Delta_k - \Delta_{k+1} - f_{k+2})^+ & \text{si} \quad \theta > \Delta_k + \Delta_{k+1} \end{cases} \tag{5}$$

The explanation of (5) is as follows. If $\theta \leq \Delta_k + \Delta_{k+1}$, the starting time of task a_{k+2} is not modified, as well as the starting times of the following tasks. If $\theta > \Delta_k + \Delta_{k+1}$, then a_{k+2} is delayed by $\theta - \Delta_k - \Delta_{k+1}$. Thus, if $\theta - \Delta_k - \Delta_{k+1} \leq f_{k+2}$, the due date of a_{k+2} is not violated and the increase of C_n resulting from a_{k+2} is equal to zero, otherwise the increase of C_n is $\theta - \Delta_k - \Delta_{k+1} - f_{k+2}$. This completes the explanation of (5).

The first task that is not delayed is the task a_{k+n_k+1}, where n_k is the smallest integer such that $\sum_{s=0}^{n_k} \Delta_{k+s} \geq \theta$. The increase of C_n is equal to zero if $\theta \leq \Delta_k$. Finally, the increase of the criterion C_n is :

$$L_k = \begin{cases} 0 & \text{if} \quad \Delta_k \geq \theta \\ \sum_{s=1}^{n_k} R_{k+s} & \text{if} \quad \Delta_k < \theta \end{cases} \tag{6}$$

Or :

$$L_k = \begin{cases} 0 & \text{if} \quad \Delta_k \geq \theta \\ \sum_{s=1}^{n_k} (\theta - \sum_{p=0}^{s-1} \Delta_{k+p} - f_{k+s})^+ & \text{if} \quad \Delta_k < \theta \end{cases} \tag{7}$$

Thus, we are able to compute the increase of C_n when we know a_k, that is the task after which A starts and n_k, that is the number of tasks that will be postponed after inserting A. Indeed, due to the "real-time" constraint, it is impossible to try all the feasable locations of A in real-time in order to keep the best one. Thus we have to find a way to compute (or evaluate) L_k that is compatible with the real-time constraint. In the next section, we propose two approaches to reach this goal.

4 Use of the basic relations.

We suppose a random task A arises, we know its duration θ and its due date D. This due date cannot be violated. We have to decide, in real-time, to start A just after one of the task a_i already scheduled, the goal being to increase as less as possible the criterion C_n (see 3) associated with the initial schedule. A huge amount of computation is required to reach this decision. In this section, we present two ways to perform most of the computation off-line, using the information related to the initial schedule, and to perform only a reasonable amount of computation in real time.

4.1 First approach.

The first approach will lead to algorithm $TR1$ presented in section 5. $TR1$ will lead to solutions close to the optimum, but the reader will see that the real-time computation is quite heavy since it requires testing several positions of task A.

4.1.1 The reference tables.

The off-line computation consists in designing so-called reference tables to decide where to insert task A. Consider the initial schedule presented in table 1. The due dates are not represented in the figure.

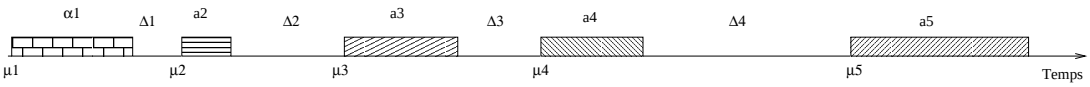

Figure 1 : A schedule example.

Task A, defined by θ and D, arises at time 0. Consider, for instance, the starting time μ_4 of task a_4. If $\theta \leq \mu_4 - (\mu_3 + \theta_3)$, then starting A after a_3 guaranties that A ends at the latest at time μ_4. On the other hand, if $\mu_4 - (\mu_3 + \theta_3) < \theta \leq \mu_4 - (\mu_2 + \theta_2)$, then we have to insert task A just after task a_2 to be sure that it will be completed by time μ_4. Similary, if $\mu_4 - (\mu_2 + \theta_2) < \theta \leq \mu_4 - (\mu_1 + \theta_1)$, we have to start task A as soon as task a_1 ends to complete it by time μ_4. Finally, if $\theta > \mu_4 - (\mu_1 + \theta_1)$, it is impossible to complete A before μ_4.
Indeed :

- if $\theta \leq \Delta_3$, the initial schedule remains unchanged if we insert task A after task a_3.
- if $\Delta_3 < \theta \leq \Delta_3 + \Delta_2$, task a_3 will be delayed by $\theta - \Delta_2$ when we start A as soon as a_2 is completed.,
- if $\Delta_3 + \Delta_2 < \theta \leq \Delta_3 + \Delta_2 + \Delta_1$, task a_2 will be delayed by $\theta - \Delta_1$ and task a_3 will be delayed by $\theta - \Delta_1 - \Delta_2$ if we insert task A after task a_1.

Thus, if we know the starting time μ_k of a_k at which A should be completed, it is possible to provide the list of the tasks after which we should insert task A to complete it on time.

The initial table X is derived from the previous remarks. X has n columns and n rows. Let us consider row i and column j.

The corresponding elements x_{ij} of X is defined as follows :

$$x_{ij} = \begin{cases} 0 & \text{if} \quad j \leq i \\ \sum_{k=i}^{j-1} \Delta_k & \text{if} \quad j > i \end{cases} \quad \text{for } i = 1, 2, \ldots, n. \tag{8}$$

Element x_{ij} is the maximal value of θ that results in postponing only tasks $a_{i+1}, \ldots, a_{j-1}$ if A is inserted just after a_i.

The initial table that corresponds to the schedule presented in figure 1 is table 1.

	μ_1	μ_2	μ_3	μ_4	μ_5
a_1	0	Δ_1	$\Delta_1 + \Delta_2$	$\Delta_1 + \Delta_2 + \Delta_3$	$\Delta_1 + \Delta_2 + \Delta_3 + \Delta_4$
a_2	0	0	Δ_2	$\Delta_2 + \Delta_3$	$\Delta_2 + \Delta_3 + \Delta_4$
a_3	0	0	0	Δ_3	$\Delta_3 + \Delta_4$
a_4	0	0	0	0	Δ_4

Table 1: Reference table related to the schedule of figure 1.

From this initial table, we derive the reference table 1, denoted by Y, and whose elements y_{ij} are defined as follows :

$$y_{ij} = \begin{cases} 0 & \text{if} \quad j \leq i \\ x_{ij} - f_j & \text{if} \quad j > i \end{cases} \quad \text{for } i = 1, 2, \ldots, n. \tag{9}$$

Reference table 2 will help to compute the increase of the criterion when inserting A. We also introduce reference table 2, denoted by V, whose elements v_{ij} are defined as follows :

$$v_{ij} = \begin{cases} 0 & \text{if} \quad j \leq i \\ \mu_j - (\mu_i + \theta_i) & \text{if} \quad j > i \end{cases} \quad \text{for } i = 1, 2, \ldots, n. \tag{10}$$

For $j > i$, table V provides the maximal value of the duration θ of task T such that T can be completed before μ_j if it starts as soon as a_i is completed.

The reference tables only depend on the initial schedule. As a consequence, they can be computed off-line. Since only the triangular upper part of the reference tables is of interest, we have to compute $(n-1)(n-2)/2$ values for each table. For a large value of n (greater than one hundred for instance) we may obtain tables that are too large to be handled in real-time. Thus, we decided to restrict the number of rows of X, Y and V to Z. The value of Z will depends on the problem at hand : it is linked to the minimal number of consecutive Δ_i periods whose sum is greater than θ. For Z, the number of useful elements of table X is $Z(2n-Z-1)/2$.

4.1.2 Use of the reference table.

Assume that a task A arises at time 0 and that $D = \mu_k$. Then the feasible locations of A, that is the locations that guaranty that A will be completed by $D = \mu_k$, are defined by keeping, in the column k of V, the indexes $i = 1,2,\ldots,p$ such that $v_{ik} \geq \theta$. We set $I_k = \{1,2,\ldots,p\}$. Then for any $i \in I_k$, we can apply relation (7) to L_i, with $n_i = k - i$:

$$L_i = \sum_{s=1}^{n_i-1} (\theta - \sum_{p=0}^{s-1} \Delta_{i+p} - f_{i+s})^+ \tag{11}$$

We always use the same convention for the sums, that is $\sum_{i=n_1}^{n_2} . = 0$ if $n_2 < n_1$. Using the elements of X, (11) becomes :

$$L_i = \sum_{s=1}^{n_i-1} (\theta - x_{i,i+s} - f_{i+s})^+ \text{ for each } i \in I_k \tag{12}$$

Thus, using the elements of table Y,

$$L_i = \sum_{s=1}^{n_i-1} (\theta - y_{i,i+s})^+ \text{ for each } i \in I_k \tag{13}$$

The previous process applies not only for $D = \mu_k$, but also for $D \in [\mu_k, \mu_k + t_k]$. If $D \in [\mu_k + t_k, \mu_{k+1})$, then we can apply the same process with $\mu_{k+1} = D$ and $\Delta_k = D - (\mu_k + t_k)$. In the algorithm presented in section 5, we do not consider this case, and we set $D = \mu_k$ for any $D \in [\mu_k, \mu_{k+1})$: it is why our approach is heuristic. Furthermore, if $I_k = \emptyset$, A cannot be completed by time μ_k.

Indeed, the fact that the number of rows of the reference table X is limited to Z may lead to discard some feasible solutions that may include the optimal solution.

4.2 Second approach.

This second approach is very simple. It consists in computing, for $k = 1,2,\ldots,n$, T_k which is the maximum of the sum of k consecutives periods. In other words :

$$T_k = \max_{i=1,2,\ldots,n-k+1} \sum_{s=0}^{k-1} \Delta_{i+s} \tag{14}$$

The first reference table, applied to the exemple represented in figure 1, is table 2.

	μ_1	μ_2	μ_3	μ_4	μ_5
a_1	0	$\Delta_1 + f_2$	$\Delta_1 + \Delta_2 + f_3$	$\Delta_1 + \Delta_2 + \Delta_3 + f_4$	$\Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + f_5$
a_2	0	0	$\Delta_2 + f_3$	$\Delta_2 + \Delta_3 + f_4$	$\Delta_2 + \Delta_3 + \Delta_4 + f_5$
a_3	0	0	0	$\Delta_3 + f_4$	$\Delta_3 + \Delta_4 + f_5$
a_4	0	0	0	0	$\Delta_4 + f_5$

Table 2: Reference table 1 : Y

V is as follows for the exemple presented in figure 1.

	μ_1	μ_2	μ_3	μ_4	μ_5
a_1	0	$\mu_2 - (\mu_1 + \theta_1)$	$\mu_3 - (\mu_1 + \theta_1)$	$\mu_4 - (\mu_1 + \theta_1)$	$\mu_5 - (\mu_1 + \theta_1)$
a_2	0	0	$\mu_3 - (\mu_2 + \theta_1)$	$\mu_4 - (\mu_2 + \theta_2)$	$\mu_5 - (\mu_2 + \theta_2)$
a_3	0	0	0	$\mu_4 - (\mu_3 + \theta_3)$	$\mu_5 - (\mu_3 + \theta_3)$
a_4	0	0	0	0	$\mu_5 - (\mu_4 + \theta_4)$

Table 3: Reference table 2 : V

We denote by $i^*(k)$ the index i that leads to T_k. The idea behind this second approach is to define the minimal value of k such that $T_k > \theta$ when the random task A arises. We denote this value by k^*.

Two cases should be considered :

1. $i^*(k^*)$ is such that :
   $\mu_{i^*(k^*)} + t_{i^*(k^*)} + \theta \leq D$. It means that it is possible to complete task A by time D if we start A as soon as task $i^*(k^*)$ is completed. In this case , we insert A after $i^*(k^*)$

2. Si $i^*(k^*)$ is such that :
   $\mu_{i^*(k^*)} + t_{i^*(k^*)} + \theta > D$. In this case, we insert A after task 1 if $\mu_1 + t_1 + \theta \leq D$, otherwise the problem has no solution.

As we can see, this approach does not take into account the so-called flexibility of the tasks : the goal is only to integrate A at a place that disburbs a minimal number of scheduled tasks. If this position leads to violate the deadline D, then we try to insert A just after the first task.

The goal of this approach is to reduce as much as possible the number of tasks of the initial schedule that should be adjusted when inserting A.

This approach does not take into account the flexibility of the tasks. Its goal is only to insert task A after a task such as to perturb as less scheduled tasks as possible. If doing so leads to violate the deadline D, then we try to insert A just after the first task. If D is still violated, then the set of feasable solutions is empty. The fact that a small number of scheduled tasks are postponed simplifies the adjustement of the existing schedule when inserting task A.

5 The heuristic algorithms.

5.1 The $TR1$ algorithm.

This algorithm is derived from the first approach. The computation can be clearly divided in two parts : the first part includes the computation that can be performed off-line, that is as soon as the schedule is known and before the random task A arises, while the second part of the computation starts when D and θ are known, that is when A arises.

Remark : Tables X, Y and V are defined with regard to the starting time of task a_1. When time roll by, task a_1 becomes a task of the past and the first task to perform task a_2, and so on. Thus, it is necessary to adjust X and Y periodically. A simple way to do that is to :

- Keep this tables constant between the starting times of any two consecutives tasks.
- Adjust table X at the end of each period by substracting Δ_i from each element of line i for $i = 1, 2, \ldots, n$.
- Recompute table Y starting from the new table X.
- Table V is adjusted by removing the first row and the first column of V and possibly completing the initial schedule.

These adjustements are not considered in this paper.

Algorithm $TR1$

1. Part 1 : Off-line computation.
   (a) Computation of table X by means of equations (8)
   (b) Computation of table V by means of equations (10)
   (c) Computation of table Y by means of equations (9)
2. Partie 2 : Real-time computation.
   (a) Define μ_k such that $D \in [\mu_k, \mu_{k+1})$.
   (b) Set $I_k = \emptyset$
   (c) For $i = 1, 2, \ldots$ do :
   If $v_{i,k} \geq \theta$ then $I_k = I_k + \{i\}$, else go to $2.(d)$
   (d) If $I_k = \emptyset$, no solution. END.
   (e) For any $i \in I_k$, compute L_i (see (13))
   (f) Compute i^* such as $L_{i^*} = \max_{i \in I_k} L_i$
   (g) Insert A after a_{i^*}

5.2 The $TR2$ algorithm.

Algorithm $TR2$ is derived from the second approach presented in section 4. It is also divised in two parts : the off-line part and the real-time part.

Algorithm $TR2$

1. Part 1 : Off-line computation.
   (a) For $k = 1, 2, \ldots, n$, compute T_k and $i^*(k)$. T_k is computed using equalities (13)
2. Partie 2 : Real-time computation.
   (a) For $k = 1, 2, \ldots, n$, do
      i. If $T_k < \theta$, go to $2.(b)$
      ii. If $T_k \geq \theta$, keep $i^*(k)$ and go to $2.(d)$
   (b) End of loop k k
   (c) No solution : END.
   (d) If $\mu_{i^*(k)} + t_{i^*(k)} + \theta \leq D$, we set task A just after task $i^*(k)$. END.
   (e) If $\mu_{i^*(k)} + t_{i^*(k)} + \theta > D$
      i. If $\mu_1 + t_1 + \theta \leq D$, we set task A just after task 1. END.
      ii. Ifai.p $\mu_1 + t_1 + \theta > D$, no solution. END.

6 The simulation software.

6.1 Algorithm.

The simulation software algorithm is as follows :

1. For θ = 20 to 100 ,step 20, do :
2. (a) For i = 1 to 400 do :
   (b) i. Build a schedule.
      ii. Generate D : we generate it after time 700 and before the beginning of the last scheduled task , using a random number.
      iii. Find the optimal insertion of (θ, D) (complete exploration).
      iv. Apply an heuristic.

Each schedule is built randomly. We generate 35 tasks that might be separated by an idle time period.

- Idle periods are only featured by their duration : they start at the end of the previous task, and stops at the beginning of the next one. This time is computed by applying the formulae : $T = 15 + 10 * (12 * \text{random}(1.0) - 6)^+$ where $(x)^+ = \max(0,x)$; $random(1.0)$ is a floating point random number that is valued between 0 and 1.
- Each of the tasks are featured with a release time, a busy time, and a deadline.
  1. The release time of the first task is 0. The release times of the next tasks are obtained by adding the busy times of the previously scheduled tasks, and the previous idle times.
  2. The busy time of a task is a random integer, randomly generated between 1 and 50.
  3. The deadline of a task is the sum of the release date of the task, the operation time of the task, and of a random integer. This random integer is generated between -50 and 50. Only 20% of these integers are less than 0.

6.2 Complexity

6.2.1 Algorithm $TR1$.

We compute the complexity of algorithm $TR1$. We begin the computation by evaluating its off-line part. We give the algorithms that corresponds to equations (8), (9) et (10) :

Computation of X :

1. For i = 1, *nbtask* do
   (a) For j = 1, i do $X[i][j] = 0$;
   (b) For $j = i$+1, *nbtask* do $X[i][j] = X[i][j\text{-}1] + \Delta_{j-1}$

The complexity of the computation of X is :

$$C_X = \sum_{i=1}^{n} \{ \sum_{j=1}^{i} 1 + \sum_{j=i+1}^{n} 2 \}$$

Computation of Y :

1. For i = 1, *nbtask* do
   (a) For j = 1, i do $Y[i][j] = 0$;
   (b) For $j = i$+1, *nbtask* do $Y[i][j] = X[i][j\text{-}1] + f_{j-1}$

The complexity of the computation of Y is :

$$C_Y = C_X = \sum_{i=1}^{n} \{ \sum_{j=1}^{i} 1 + \sum_{j=i+1}^{n} 2 \}$$

Computation of V :

1. For i = 1, *nbtask* do
   (a) For j = 1, i do $V[i][j] = 0$;
   (b) For $j = i$+1, *nbtask* do $V[i][j] = \mu_j - \mu_i - \theta_i$

The complexity of the computation of V is :

$$C_V = \sum_{i=1}^{n} \{ \sum_{j=1}^{i} 1 + \sum_{j=i+1}^{n} 4 \}$$

Then, we sum $C_X + C_Y + C_V$ to obtain the off-line complexity :

$$\begin{aligned} C_{hors-ligne} &= 2C_X + C_V \\ &= 2 * \{\sum_{i=1}^{n}\{\sum_{j=1}^{i} 1 + \sum_{j=i+1}^{n} 2\}\} + \sum_{i=1}^{n}\{\sum_{j=1}^{i} 1 + \sum_{j=i+1}^{n} 4\} \\ &= 2 * \{\frac{3n^2 - n}{2}\} + \frac{5n^2 - 3n}{2} \\ &\leq 11n^2 \end{aligned}$$

Let's look the real-time part of the algorithm, line per line :

- line 2.a : up to n operations
- line 2.b : 1 operation
- line 2.c : up to n times 3 operations (test, addition, affectation).
- line 2.e : up to n applications of the formulae (13), which uses up to $3 * (\mathrm{n}-2)$ operations.
- line 2.f : up to n operations.

Finally :

$$C_{on-line} = n + 1 + 3n + \sum_{i=1}^{n}\sum_{j=1}^{n-2} 3 + n = 3n^2 - n + 1 \leq 3n^2$$

6.2.2 Algorithm $TR2$.

We compute the complexity of algorithm $TR2$. Firstly, we evaluate the off-line process. From equation (14), we deduce the algorithm :

1. For i = 2, $nbtask$ do $T[1][i] = \Delta_{i-1}$;
2. For i = 2, $nbtask - 1$ do
   (a) For i = i, $nbtask$ do $T[i][j] = T[i-1][j] + \Delta_{j-i}$

The number of operations to execute is :

$$C_{off-line} = \sum_{i=1}^{n} 1 + \sum_{i=2}^{n-1}\sum_{j=i}^{n} 2 = n^2 - 3 \leq n^2$$

Now, let's evaluate the on-line part of the algorithm :

- line 2.a : up to n operations.
- remains : up to 5 operations.

Finally :

$$C_{on-line} = \sum_{i=1}^{n} 1 + 5 = n + 5 \leq 2n$$

6.3 Numerical results.

We have tested two thousands examples with each algorithm $TR1$ and $TR2$. In all these exemples, the optimal insertion was found using a complete exploration of the possible solutions.

In table 4, we provide for each heuristic :

- the percentage of examples that leads to an optimal solution,
- the percentage of examples that leads to solutions that may cost up to 10% more than the optimal solution,
- the percentage of examples that leads to solutions that may cost up to 15% more than the optimal solution.

	0%	$\leq 10\%$	$\leq 15\%$
$TR1$	92%	95%	97%
$TR2$	84.5%	84.7%	89.6%

Table 4: Results

7 Conclusion.

The goal of this paper was to insert a lethal deadlined random task in a given schedule, while minimizing the increase of the value of the criterion that is the sum of the delays of the previously scheduled tasks. This goal should be reached in real-time.

Our future research objective is twofold :

- Analyse the problem in which the deadline of the random task is no more lethal, but strongly penalized.
- Analyse the problem in which the random task is composed with series of subtasks that are separated by periods. The periods may vary beetween strongly constrainted bounds.

References

[1] MOK A. *Fundamental Design Problems of Distributed Systems for the hard Real-Time Environment*. PhD thesis, MIT Laboratory for Computer Sciences, 1983.

[2] LEVNER E. CHAUVET F. and PROTH J.M. On-line part scheduling in a surface traitment system. *Journal of Operational Research, IJOR,*, vol. 120/2, January 2000.

[3] LIU C.L. and LAYLAND J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, vol. 20, n°1, pp. 46-61, 1973.

[4] CHAUVET F. et PROTH J.M. On-line scheduling in assembly processes. *Information Systems and Operation Research*, vol.39 n°1, feb 2001.

[5] PROTH J.M. *On-line Scheduling in Supply Chain Environment*, chapter Optimal Control and Partial Differential Equations. Eds. Menaldi et al., IOS Press, 2001.

[6] LEHOCZKY J.P. SHA L. and DING Y. The rate monotonic scheduling algorith. exact characterization and average case behavior. *IEEE Real-time Systems Symposium*, 1989.

[7] DERTOUZOS M. Control robotics : the procedural control of physical processors. *Proceedings of the IFIP Congress*, pp. 807-813, 1974.

[8] RAMAMRITHAM K. ZHAO W. and STANKOVIC A. Preemptive scheduling under time and resource constraints. *IEEE Transaction on Computers*, vol. 36, n°8, 1987.

Contents

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399